МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий Кафедра Программной инженерии Специальность 1-40 01 01 Программное обеспечение информационных технологий   
Направление специальности 1-40 05 01 Информационные системы и технологии

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»   
Тема «Программное средство для автошкол»

Исполнитель

студент (ка) 2 курса группы 6 Сачишин Иван Александрович

(Ф.И.О.)

Руководитель работы асс. Радиванович Д.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Пацей Н.В.

(подпись)

Минск 2021

Оглавление

[ВВЕДЕНИЕ 3](#_Toc73174086)

[1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4](#_Toc73174087)

[2 Моделирование предметной области и разработка функциональных требований. 6](#_Toc73174088)

[2.1 Описание инструментов для разработки программного средства 6](#_Toc73174089)

[2.2 Описание функциональности программного средства 8](#_Toc73174090)

[3 Проектирование программного средства 10](#_Toc73174091)

[3.1 Модель базы данных 10](#_Toc73174092)

[3.2 Структура проекта 12](#_Toc73174093)

[3.3 Структура классов программного средства 14](#_Toc73174094)

[3.4 Работа с базой данных 15](#_Toc73174095)

[3.5 Аутентификация пользователей в системе 16](#_Toc73174096)

[3.6 Окна приложения 16](#_Toc73174097)

[4 Реализация программного средства 18](#_Toc73174098)

[4.1 Реализация работы с базой данных 18](#_Toc73174099)

[4.2 Реализация регистрации и аутентификации пользователей 18](#_Toc73174100)

[4.3 Реализация основных функций программного средства 21](#_Toc73174101)

[5 Тестирование, проверка работоспособности и анализ полученных данных 25](#_Toc73174102)

[**Руководство по установке и использованию** 29](#_Toc73174103)

[ЗАКЛЮЧЕНИЕ 31](#_Toc73174104)

[СПИСОК ЛИТЕРАТУРЫ 32](#_Toc73174105)

[ПРИЛОЖЕНИЕ 1 33](#_Toc73174106)

[ПРИЛОЖЕНИЕ 2 33](#_Toc73174107)

[ПРИЛОЖЕНИЕ 3 34](#_Toc73174108)

# ВВЕДЕНИЕ

В наше время автомобили стали неотъемлемой частью нашей жизни, это уже не роскошь, как раньше, а средство передвижения, которое есть почти в каждой семье. Они позволяют преодолевать огромные расстояния за время, в разы меньшее чем на общественном транспорте. Очевидно, что такая вещь, как автомобиль, является источником повышенной опасности и требует специфических навыков и знаний, которые нужны чтобы управлять им. Такие навыки и знания приобретаются в автошколах. В процессе обучения проводятся теоретические и практические занятия на автодроме и в городе. Каждый курсант и автоинструктор в автошколе сталкивались с проблемой планирования занятий. Как правило, занятия планируются по телефону в некомфортных для обоих условиях. Позже это может привести к тому, что один из участников звонит второму, чтобы перенести занятие так как он при планировании не учел что выбранное время ему не подходит или, например, инструктор может забыть или потерять свои записи, о запланированном занятии, если он пользуется бумажными носителями информации.

Целью данного курсового проекта является разработка программного средства, которое позволит курсанту автошколы в любое время в комфортных для него условиях, быстро и удобно, планировать свои занятия с инструкторами. Автоинструктору – легко просматривать своё расписание занятий и информацию, связанную с предстоящим занятием. Также программное средство позволяет администратору просматривать информацию о пользователях зарегистрированных в системе, о всех занятиях и управлять необходимой для функционирования системы информацией.

# Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству

В данном курсовом проекте будут использоваться дополнительные материалы из таких источников, как <https://metanit.com/> [1] для получения дополнительной информации о WPF, <https://refactoring.guru/> [2] для дополнительной информации о внедрении паттернов проектирования в программное средство с целью создания более гибкого продукта, а также официальный сайт Microsoft <https://docs.microsoft.com/> [3] для получения общей информации о C# и Microsoft SQL Server.

В качестве аналога был выбрано программное средство [Автошкола-контроль](http://dscontrol.ru/onlinelearning/)[4].

[Автошкола-контроль](http://dscontrol.ru/onlinelearning/)

Частью функционала данного средства является онлайн запись на практические занятия с автоинструктором.

Для клиентов страница записи содержит таблицу со свободными датами проведения занятий и интервалами времени(рис. 1.1):

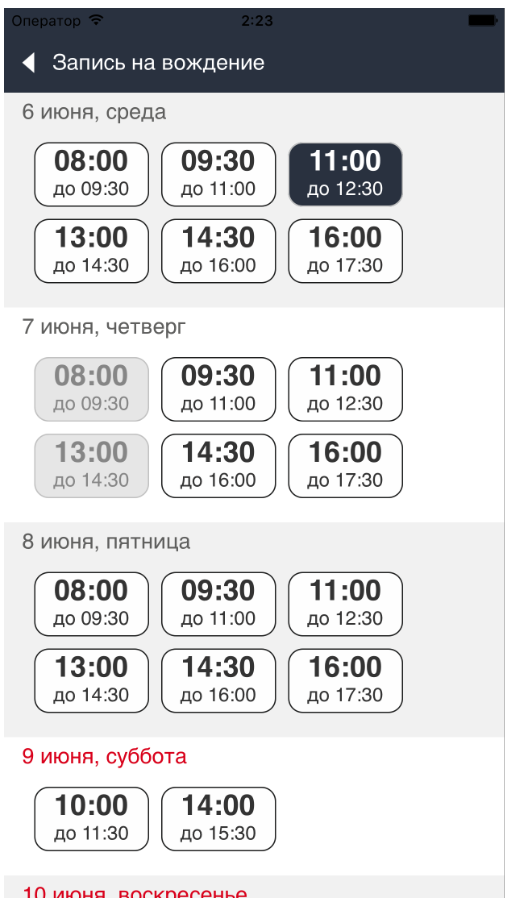


Рис. 1.1 – Планирование времени проведения занятия

После выбора даты и времени занятия нужно подтвердить бронирование времени (рис. 1.2):

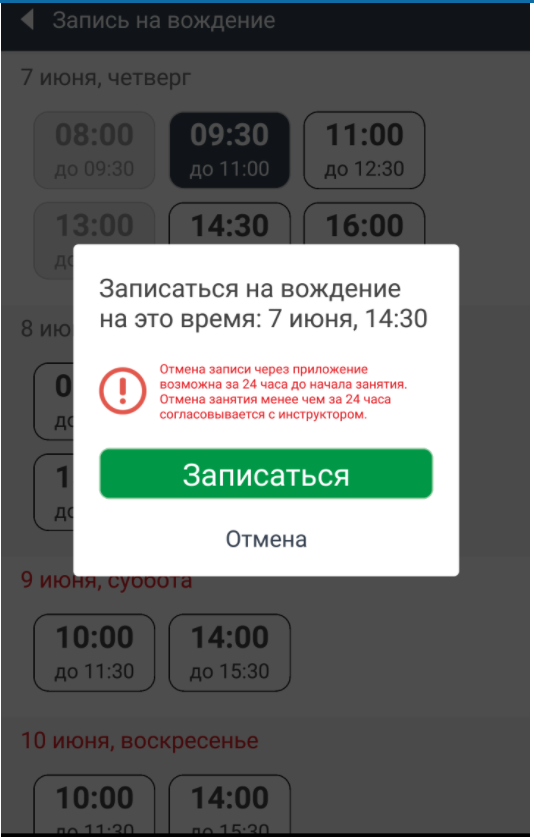


Рис. 1.2 – Подтверждение записи

После подтверждения записи на занятие можно просматривать своё расписание в специальной таблице.

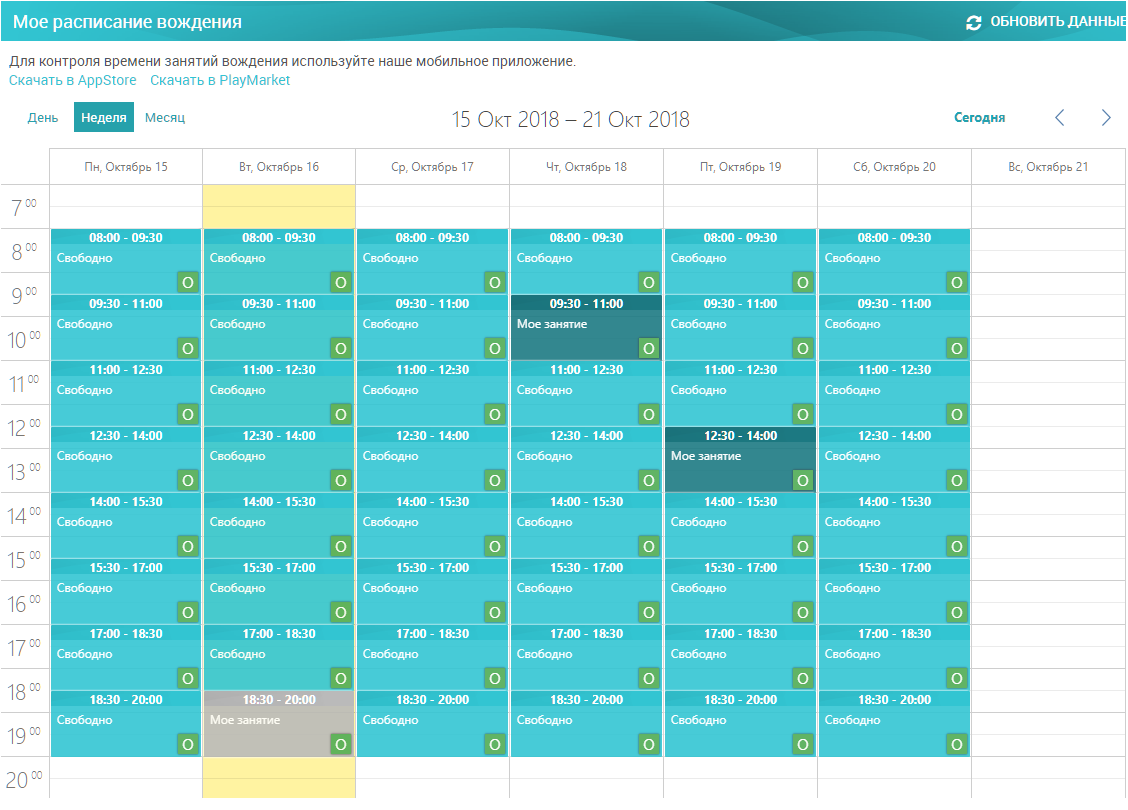
Таблица просмотра запланированных занятий (рис 1.3):

Рис. 1.3 – Таблица занятий

Исходя из анализа аналога, были выявлены сильные и слабые стороны обоих проектов.

Сильные стороны:

* Удобный механизм записи на занятия и просмотра запланированных занятий;
* Возможность просматривать профили инструкторов;

Слабые стороны:

* Перегруженный дизайн [Автошкола-контроль](https://dscontrol.ru/online-recording/);

Исходя из анализа аналогов, были выделены требования к программному обеспечению:

* Удобный и понятный интерфейс для пользователей с любой компьютерной грамотностью;
* Разделение бизнес-логики приложения на клиентскую часть и часть администратора;

# Моделирование предметной области и разработка функциональных требований.

# Описание инструментов для разработки программного средства

В ходе разработки данного программного средства используются следующие инструменты:

* Объектно-ориентированный язык программирования C#;
* Платформа для кроссплатформенной разработки с открытым исходным кодом .NET Framework;
* Расширяемый язык разметки XAML;
* Система управления базами данных Microsoft SQL Server;
* Шаблон проектирования MVVM;
* Фреймворк для работы с базой данных Entity Framework 6;
* Технология WCF;
* Интегрированная среда разработки *Microsoft Visual Studio* 2019;

C# [5]— это язык программирования, предназначенный для разработки самых разнообразных приложений, предназначенных для выполнения в среде .NET Framework. Visual C# — это реализация языка C# корпорацией Майкрософт. Поддержка Visual C# в Visual Studio обеспечивается с помощью полнофункционального редактора кода, компилятора, шаблонов проектов, конструкторов, мастеров кода, мощного и удобного отладчика и многих других средств. Библиотека классов .NET Framework предоставляет доступ ко многим службам операционной системы и другим полезным, правильным классам, что существенно ускоряет цикл разработки.

WCF (Windows Communication Foundation) – программный фреймворк для обмена данными между приложениями, входящий в состав .NET FRAMEWORK.

XAML — это декларативный язык разметки. С точки зрения модели программирования .NET Core язык XAML упрощает создание пользовательского интерфейса для приложения .NET Core. Можно создать видимые элементы пользовательского интерфейса в декларативной XAML-разметке, а затем отделить определение пользовательского интерфейса от логики времени выполнения, используя файлы кода программной части, присоединенные к разметке с помощью определений разделяемых классов. Язык XAML напрямую представляет создание экземпляров объектов в конкретном наборе резервных типов, определенных в сборках.

Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

SQL Server — это основа платформы обработки данных Майкрософт, которая предоставляет надежную и устойчивую производительность (в том числе благодаря технологиям обработки данных в памяти) и помогает быстрее извлечь ценную информацию из любых данных, расположенных как в локальной среде, так и в облаке.

MVVM — шаблон проектирования архитектуры приложения, который позволяет отделить логику приложения от визуальной части, что упрощает тестирование и поддержку приложения.

# Описание функциональности программного средства

На основе анализа были составлены следующие функциональные требования для программного средства для клиента:

* Просмотр расписания и запись на занятия;
* Просмотр своих занятий и их отмена;
* Просмотр профилей инструкторов и оставление отзывов о них;
* Просмотр своего профиля и его изменение;

Для администратора:

* Просмотр всех занятий и информации о них;
* Добавление инструкторов;
* Просмотр профилей пользователей и их удаление.
* Редактирование автопарка

Для автоинструктора:

* Просмотр своих занятий и информации о них;
* Просмотр своего профиля и его редактирование.

Далее приведена схема Use-case, описывающая функциональность программного средства для каждого из пользователей (рис 2.1):

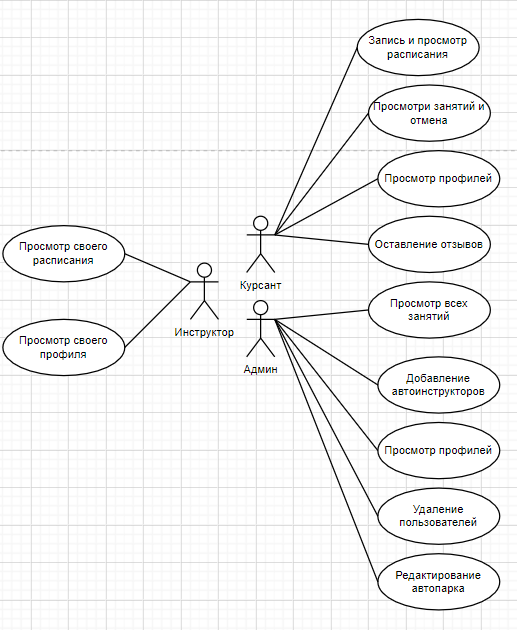


Рисунок 2.1 – Диаграмма деятельностей

Из данной схемы видно, что клиент может выполнять операции по писку свободных занятий, их бронировании и изменений своего профиля, инструктор в свою очередь просматривает расписание и редактирует свой профиль, а администратор может изменять автопарк, просматривать и удалять пользователей, а также добавлять и удалять инструкторов.

# Проектирование программного средства

# Модель базы данных

Для создания программного средства была разработана база данных DriverPlanner\_DB, состоящая из 9 таблиц. Описания таблиц базы данных представлены в таблицах 3.1 – 3.9:

Таблица 3.1 – таблица Genders

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| GenderName | nvarchar(128) | Содержит название пола |
| GenderDescription | nvarchar(MAX) | Содержит полное описание пола |

Таблица 3.2 – таблица Cars

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| CarID | int | Содержит идентификатор машины |
| CarName | nvarchar(20) | Содержит название марки машины |
| ImageIndex | int | Содержит индекс картинки |
| CategoryName | nvarchar(2) | Содержит название категории машины |
| CarYear | int | Содержит год выпуска авто |

Таблица 3.3 – таблица ClassInterval

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| IntervalNumber | int | Порядковый номер интервала времени |
| TimeInterval | nvarchar(MAX) | Строка с описанием интервала времени |

Таблица 3.4 – таблица Categotries

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| CategoryName | nvarchar(2) | Имя категорий авто |
| CategoryDescription | nvarchar(100) | Описание категорий авто |

Таблица 3.5 – таблица FeedBacks

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| FeedBackID | int | ID отзыва |
| FeedBackMessage | nvarchar(MAX) | Содержит сообщение в отзыве |
| UserID | int | Содержит ID пользователя |
| InstructorID | int | Содержит ID инструктора |

Таблица 3.6 – таблица Users

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| UserID | int | Идентификатор пользователя |
| FIO | nvarchar(MAX) | ФИО пользователя |
| BirthDate | datetime | Дата рождения |
| GenderName | nvarchar(128) | Пол |
| Login | nvarchar(MAX) | Логин пользователя |
| UserEMAIL | nvarchar(MAX) | Почта пользователя |
| ImageIndex | int | Индекс картинки пользователя |
| UserVK | nvarchar(MAX) | ВК пользователя |
| UserPhone | nvarchar(12) | Номер телефона |
| HashPass | nvarchar(MAX) | Хэш пароля |

Таблица 3.7 – таблица Instructors

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| InstructorID | int | ID инструктора |
| FIO | nvarchar(MAX) | ФИО инструктора |
| ImageIndex | int | Индекс фотографии |
| GenderInstructor | nvarchar(128) | Пол |
| CarID | int | ID автомобиля |
| InstructorBirth | datetime | Дата рождения |
| InstructorEMAIL | nvarchar(MAX) | Почта пользователя |
| InstructorVK | nvarchar(MAX) | ВК инструктора |
| InstructorPhone | nvarchar(MAX) | Номер телефона |
| Login | nvarchar(MAX) | Логин |
| HashPass | nvarchar(MAX) | Хэш пароля |

Таблица 3.8 – таблица TimeTable

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| ClassID | int | ID занятия |
| DateOfClass | datetime | Дата |
| IntervalCode | int | Дата рождения |
| UserID | int | Почта пользователя |
| InstructorID | int | ВК инструктора |

Таблица 3.9 – таблица Admins

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| AdminID | int | ID Администратора |
| Login | nvarchar(MAX) | Логин |
| Name | nvarchar(MAX) | Имя |
| AdminEmail | nvarchar(MAX) | Почта администратора |
| HashPass | nvarchar(MAX) | Хэш пароля |

Диаграмма базы данных, отображающая связи между таблицами, представлена на рисунке 3.1

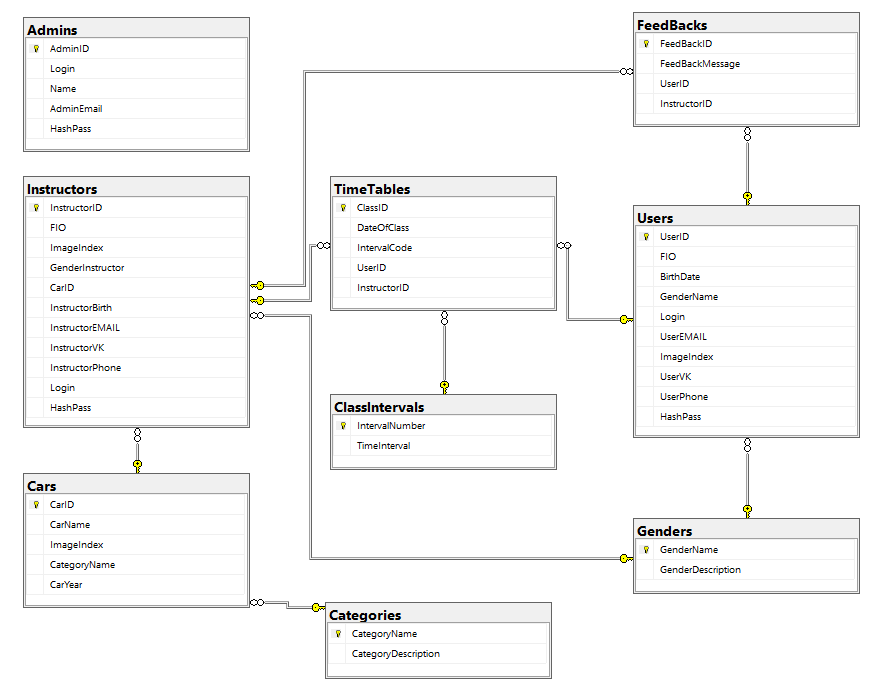


Рисунок 3.1 – Диаграмма базы данных

Ограничения целостности

# Структура проекта

Программное средство выполнено тремя проектам и состоит их клиентской части, проекта с WCF сервисом и проектом с хостом, структура которого показана на рисунке 3.2

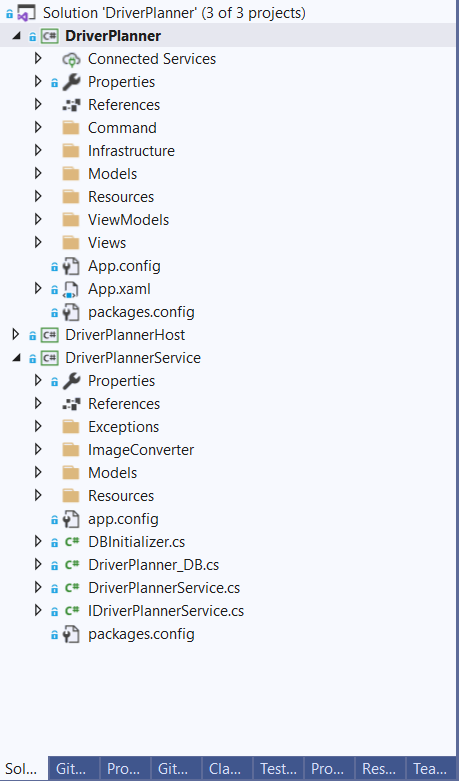


Рисунок 3.2 – Структура проекта

Таблица 3.7 – Описание структуры проекта

|  |  |
| --- | --- |
| Элемент | Описание |
| Dependencies | Содержит зависимости проекта от других сборок и библиотек |
| References | Содержит ссылки на другие проекты, сборки и т.д. |
| Connected Service | Содержит подключенный WCF сервис |
| Папка «Infrastructure» | Содержит элементы инфраструктуры проекта, такие как, конвертеры, атрибуты валидации, валидаторы, классы шифраторы пароля |
| Папка «Models» | Содержит в себе модели для хранения данных |
| Папка «Resources» | Содержит шрифты, словари ресурсов |
| Папка «ViewModels» | Содержит viewmodels для MVVM паттерна |
| Папка «Views» | Содержит представления, которые отвечают за визуальную часть для MVVM паттерна |
| Папка «Exceptions» | Содержит пользовательские исключения, генерируемые на сервисе |
| DB\_Initializer | Инициализатор базы данных, присваивает начальные данные в базу данных при создании |
| DriverPlanner\_DB | Контекст базы данных |
| DriverPlannerService | Реализация методов из IDriverPlannerService |
| IDriverPlannerService | Интерфейс, который определяет набор методов для вызова с клиента |
| App.config | Содержит строки конфигурации приложения |
| App.xaml | Содержит стартовую точку, с которой начинается выполнение приложения |

# Структура классов программного средства

Структура классов программного средства представлена в виде диаграммы классов.

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

Основными элементами являются классы и связи между ними. Классы характеризуются при помощи атрибутов и операций.

Атрибуты описывают свойства объектов класса. Большинство объектов в классе получают свою индивидуальность из-за различий в их атрибутах и взаимосвязи с другими объектами. Однако, возможны объекты с идентичными значениями атрибутов и взаимосвязей. Т.е. индивидуальность объектов определяется самим фактом их существования, а не различиями в их свойствах. Имя атрибута должно быть уникально в пределах класса. За именем атрибута может следовать его тип и значение по умолчанию [7].

В данном пункте будут рассматриваться только классы модели приложения без структуры модели представления.

Модель приложения представлена 9 основными классами.

Класс Car определят автомобиль, который имеет свой идентификатор, фото и навание, год выпуска, а так же связан с инструктором.

Класс Gender определяет пол человека. Имеет полное описание и краткое имя.

Класс Categorie определяет категорию транспортного средства и предоставляет описание категорий и краткое имя.

Класс Intervals определяет интервалы времени для занятия. Имеет порядковый номер интервала и полный интервал времени.

Класс Feedback определяет отзыв об инструкторе и имеет ссылку на курсанта и инструктора, содержит сообщение и персональный идентификатор.

Класс Users определяет курсантов и имеет ссылку на Gender, имеет ФИО,

Дату рождения, логин и хеш пароля. Наследуется от UserBase.

Класс Instructors определяет инструктора, имеет ссылку на Car и Gender. Содержит ФИО, дату рождения, логин и хеш пароля Наследуется от UserBase.

Класс Admin определяет администратора приложения. Содержит логин и хеш пароля. Наследуется от UserBase.

Класс TimeTable определяет расписание занятий. Имеет ссылку на ClassInterval, User, Instructor. Имеет поле даты занятия.

На основании данных классов была создана база данных и ее таблицы. База данных с таблицами создавалась с помощью фреймворка Entity Framework 6.

Диаграмма классов представлена на рисунке 3.3

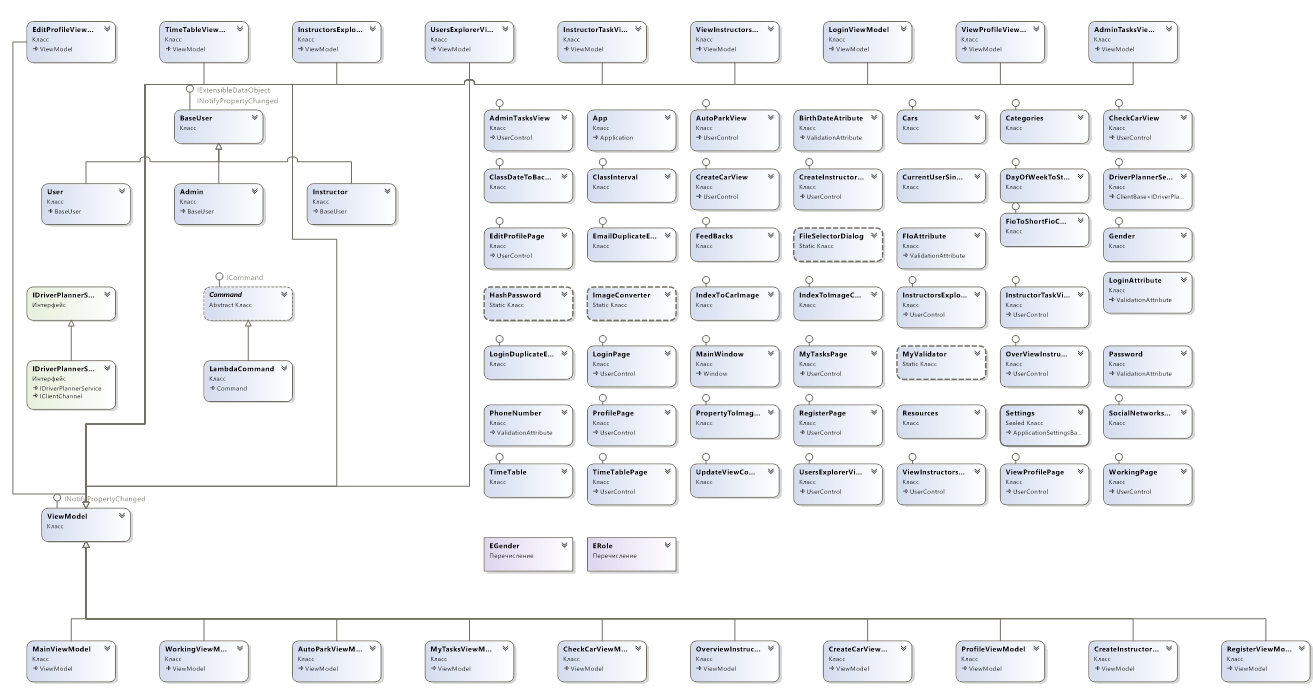


Рисунок 3.3 – Диаграмма классов ПО

# Работа с базой данных

Вся работа с базой данных целиком возложена на WCF сервис. При разработке приложения использовался Entity Framework 6. Клиент взаимодействует с сервисом посредством вызова методов сервиса с помощью хоста. При создании базы данных использовался code-first подход т.е. база данных генерируется с помощью заранее определенных моделей. В случае отсутствия базы данных, при первом обращений к ней, база будет создана и инициализирована базовыми необходимыми значениями с помощью инициализатора.

# Аутентификация пользователей в системе

Программное средство подразумевает использование несколькими пользователями с разными правами. Для этого необходима регистрация и аутентификация пользователей в системе. В системе может зарегистрироваться каждый, однако инструкторов может регистрировать только администратор. Данные, введенные пользователями, должны проходить валидацию, а пароль – хешироваться для безопасности.

Для этого разрабатывается сервис, который будет регистрировать и аутентифицировать пользователей. Схема работы алгоритма регистрации приведена в приложении 1, а схема работы алгоритма авторизации – в приложении 2.

# Окна приложения

Набор окон приложения доступных каждому пользователю, будет отличаться, т. к. отличаются их возможности в системе. Для этого необходимо разработать схему переходов для каждого пользователя. Запуск ПО будет начинаться с окна логина, откуда пользователь может перейти к регистрации, или залогиниться и перейти к основному меню. Далее он может записываться на занятия, просматривать профили и расписание и изменять свои профили и отменять занятия, а также оставлять отзывы, а администратор, в свою очередь, может управлять различными данным. Схемы пользователей представлены на рисунках 3.4-3.6.

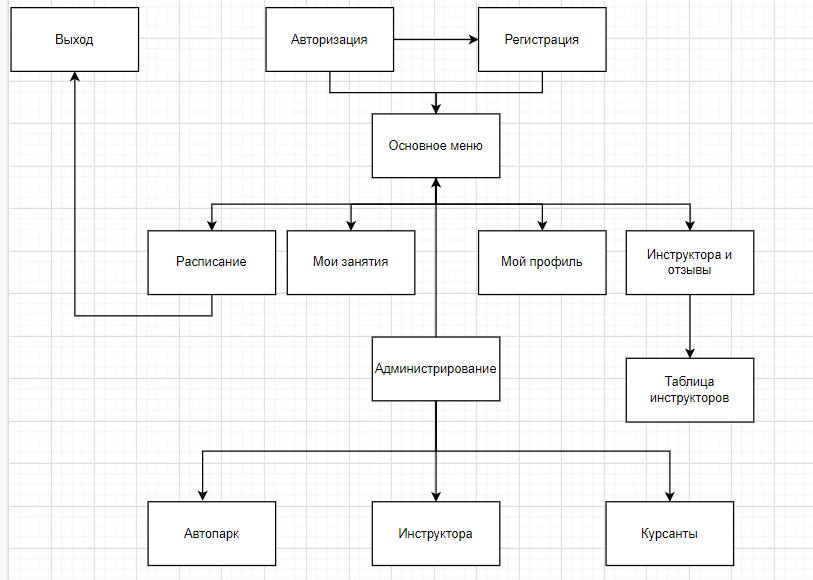


Рисунок 3.4 – Схема окон администратора

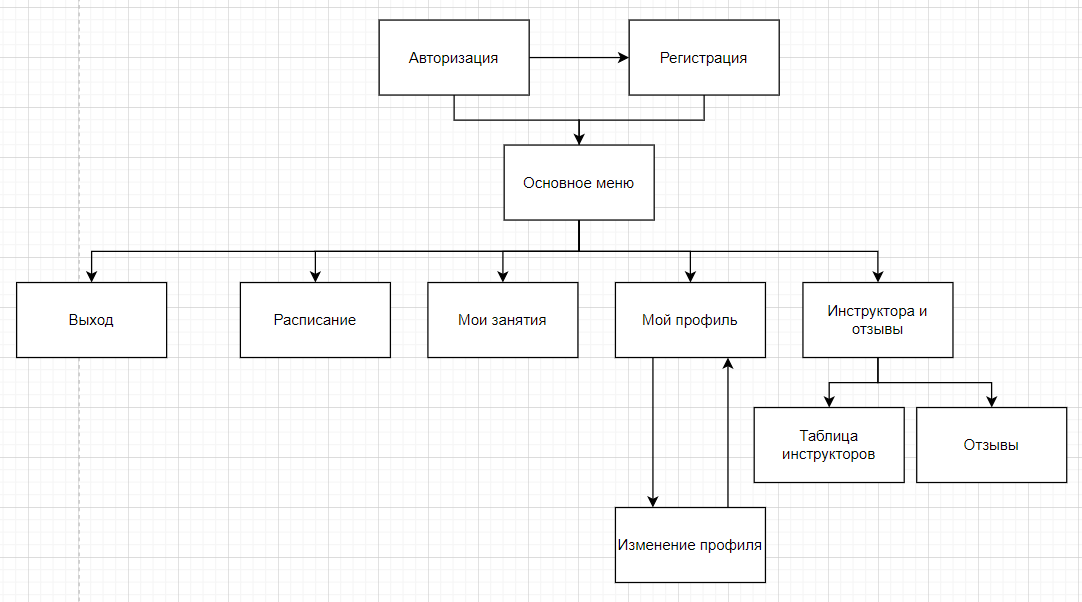


Рисунок 3.5 – Схема окон пользователя

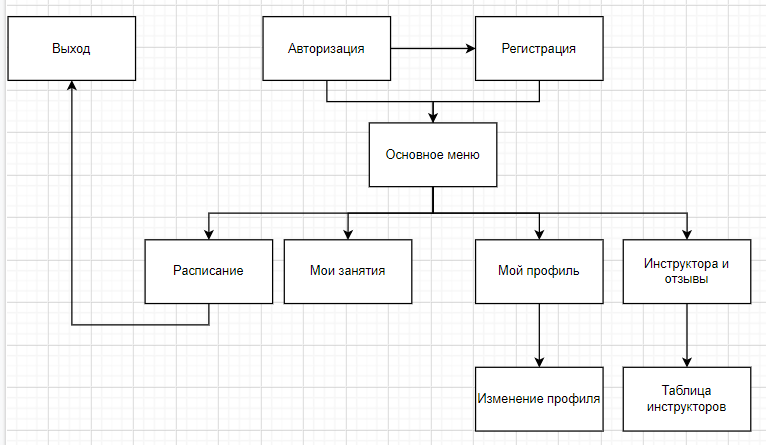


Рисунок 3.6 – Схема окон инструктора

Как можем заметить из приведенных диаграмм, самым привелегированным пользователем является администратор. Так как он может управлять аккаунтами остальных пользователей.

# Реализация программного средства

# Реализация работы с базой данных

База данных создается на основе спроектированных классов с помощью подхода Code-first. WCF сервис устанавливает связь с базой данных с помощью DbContext. Чтобы создать свою базу данных с помощью codefirst, необходимо унаследовать свой класс-контекст от dbcontext. Контекст устанавливает сеанс с помощью строки подключения, расположенной в app.connfig.

|  |
| --- |
| public class DriverPlanner\_DB : DbContext  {  public DriverPlanner\_DB() : base("DriverPlanner\_DB")  {  Database.SetInitializer(new DBInitializer());  }    public virtual DbSet<Categories> CATEGORIES { get; set; }  public virtual DbSet<ClassInterval> INTERVALS { get; set; }  public virtual DbSet<Gender> GENDERS { get; set; }  public virtual DbSet<Cars> CARS { get; set; }  public virtual DbSet<User> USERS { get; set; }  public virtual DbSet<Instructor> INSTRUCTORS { get; set; }  public virtual DbSet<Admin> ADMINS { get; set; }  public virtual DbSet<FeedBacks> FEEDBACKS { get; set; }  public virtual DbSet<TimeTable> TIMETABLE { get; set; }    } |

Листинг 4.1 – Контекст базы данных

# Реализация регистрации и аутентификации пользователей

Для регистрации и аутентификации пользователей используется сервис IDriverPlanner, который содержит в себе методы TryRegister и TryLogin для регистрации пользователя по форме и Login для его аутентификации. Интерфейс IDriverPlanner приведен в листинге 4.3.

|  |
| --- |
| public interface IDriverPlanner  {  [OperationContract]  (int, BaseUser) TryLogin(string login, string hashPasssword);  [OperationContract]  [FaultContract(typeof(EmailDuplicateException))]  [FaultContract(typeof(LoginDuplicateException))]  User TryRegister(User newUser);  [OperationContract]  (int, BaseUser) TryLogin(string login, string hashPasssword);  } |

Листинг 4.3 – Интерфейс IDriverPlanner

Т. к. в проекте используется паттерн MVVM, то все операции выполняются в модели представления (Viewmodel) определенного представления (User Control или Window). Пример взаимодействия модели, модели представления и представления рассмотрим на примере регистрации.

|  |
| --- |
| public User TryRegister(User newUser)  {  try  {  using (var ctx = new DriverPlanner\_DB())  {  #region CheckForDuplicatesOfEmailAndLogin  #region ForUserTable var resEmailInUser = ctx.USERS.Where(t => t.UserEMAIL == newUser.UserEMAIL).FirstOrDefault(); if (resEmailInUser != null) throw new FaultException<EmailDuplicateException>(new EmailDuplicateException(newUser.UserEMAIL), new FaultReason("Аккаунт с такой почтой уже зарегистрирован")); var resLoginInUser = ctx.USERS.Where(t => t.Login == newUser.Login).FirstOrDefault(); if (resLoginInUser != null) throw new FaultException<EmailDuplicateException>(new EmailDuplicateException(newUser.UserEMAIL), new FaultReason("Аккаунт с таким логином уже зарегистрирован"));  #endregion  #region ForInstructorstable var resEmailInIstructor = ctx.INSTRUCTORS.Where(t => t.InstructorEMAIL == newUser.UserEMAIL).FirstOrDefault();  if (resEmailInIstructor != null) throw new FaultException<EmailDuplicateException>(new EmailDuplicateException(newUser.UserEMAIL), new FaultReason("Аккаунт с такой почтой уже зарегистрирован"));  var resLoginInInstructor = ctx.INSTRUCTORS.Where(t => t.Login == newUser.Login).FirstOrDefault(); if (resLoginInInstructor != null) throw new FaultException<EmailDuplicateException>(new EmailDuplicateException(newUser.UserEMAIL), new FaultReason("Аккаунт с таким логином уже зарегистрирован"));  #endregion  #region ForAdmintable var resEmailInAdmin = ctx.ADMINS.Where(t => t.AdminEmail == newUser.UserEMAIL).FirstOrDefault(); if (resEmailInAdmin != null) throw new FaultException<EmailDuplicateException>(new EmailDuplicateException(newUser.UserEMAIL), new FaultReason("Аккаунт с такой почтой уже зарегистрирован")); var resLoginInAdmin = ctx.ADMINS.Where(t => t.Login == newUser.Login).FirstOrDefault(); if (resLoginInAdmin != null) throw new FaultException<EmailDuplicateException>(new EmailDuplicateException(newUser.UserEMAIL), new FaultReason("Аккаунт с таким логином уже зарегистрирован"));  #endregion  #endregion  var registeredUser = ctx.USERS.Add(newUser);  Console.WriteLine(registeredUser.FIO);  ctx.SaveChanges();  return registeredUser;  }  }  catch (Exception ex)  {    Console.WriteLine(ex.Message);  throw;  }  } |

Листинг 4.4 – Метод Register

Метод Register (листинг 4.4) регистрирует пользователя исходя из данных, введенных в форму. Здесь проверяется существование пользователя с идентичным логином и почты. Если пользователя с идентичным логином или почтой не существует, то пароль хешируется с помощью хешера MD5 и создается новый пользователь, который записывается в базу данных с помощью контекста базы данных.

Данный метод вызывается в модели представления, которая получает данные из представления. Список свойств представлен в листинге 4.5

|  |
| --- |
| [Required(ErrorMessage = "ФИО обязательно для заполнения")]  [FIo]  [MinLength(6, ErrorMessage = "Минимальная длинна ФИО - 6"), MaxLength(30, ErrorMessage = "Максимальная длинна ФИО - 30")]  public string FIO  {  get { return \_fio; }  set { Set(ref \_fio, value); }  }  private string \_login;  [Required(ErrorMessage = "Логин обязателен для заполнения")]  [Login]  [MinLength(6, ErrorMessage = "Минимальная длинна логина - 6"), MaxLength(12, ErrorMessage = "Максимальная длинна логина - 12")]  public string Login  {  get { return \_login; }  set { Set(ref \_login, value); }  }  private string \_errorMsg = "";  public string ErrorMSG  {  get { return \_errorMsg; }  set { Set(ref \_errorMsg, value); }  }  private string \_password;  [Required(ErrorMessage = "Пароль обязателен для заполнения")]  [StringLength(12, ErrorMessage = "Максимальная длинна пароля: 12 символов"), MinLength(6, ErrorMessage = "Минимальная длинна пароля: 6")]  [Password]  public string Password  {  get { return \_password; }  set { Set(ref \_password, value); }  } |

Листинг 4.5 – Свойства модели представления RegisterViewModel

Свойства модели представления проходят валидацию при помощи атрибутов валидации представленных на Листинге 4.5 и валидатора, код которого представлен на листинге 4.6

|  |
| --- |
| public static class MyValidator  {  public static string Validate(dynamic obj)  {  var results = new List<ValidationResult>();  var context = new ValidationContext(obj);  if (!Validator.TryValidateObject(obj, context, results, true))  {  foreach (var error in results)  {  return error.ErrorMessage;  }  }    return null;  }  } |

Листинг 4.6 – Валидатор

В данном примере валидатор проверяет все свойства переданного ему класса на соответствие заданным атрибутам и если ошибка будет обнаружена, то вернет результат.

В WPF используется механизм привязок, которые позволяют установить связь между полями модели представления и представления. Для этого модель представления связывается с представлением с помощью DataContext, а сами свойства привязываются с помощью конструкции Binding. Примеры привязки DataContext и свойств приведены в листингах 4.7 - 4.8.

|  |
| --- |
| <DataTemplate DataType="{x:Type viewmodels:RegisterViewModel}">  <views:AdminRegisterView/>  </DataTemplate> |

Листинг 4.7 – Привязка DataContext

|  |
| --- |
| <Button HorizontalAlignment="Left" Command="{Binding UpdateViewCommand}" Width="120" CommandParameter="Register" Content="Регистрация" Style="{StaticResource ConfirmLR}"/> |

Листинг 4.8 – Привязка свойства Command к Button

# Реализация основных функций программного средства

В приложении реализована команда навигация между окнами, которая представлена в приложении 3, это сделано с помощью элемента ContentControl. С помощью биндинга на свойство типа базоваого класса модели представления свойства source у ContentControl, мы можем динамически менять страницы по нажатию кнопки.

Также используется класс команды, который позволяет привзяывать команды к View и легко создавать новые команды, класс представлен на листинге 4.8.

|  |
| --- |
| internal class LambdaCommand : Command  {  private readonly Action<object> \_execute;  private readonly Func<object, bool> \_canExecute;  public LambdaCommand(Action<object> \_execute, Func<object, bool> \_canExecute = null)  {  this.\_execute = \_execute;  this.\_canExecute = \_canExecute;  }  public override bool CanExecute(object parameter) => \_canExecute?.Invoke(parameter) ?? true;  public override void Execute(object parameter) => \_execute(parameter);  } |

Листинг 4.9 – Класс LambdaCommand

В приложении реализованна команда просмотра расписания. При загрузке вкладки можно выбрать требуемую дату, представлено на рис. 4.10

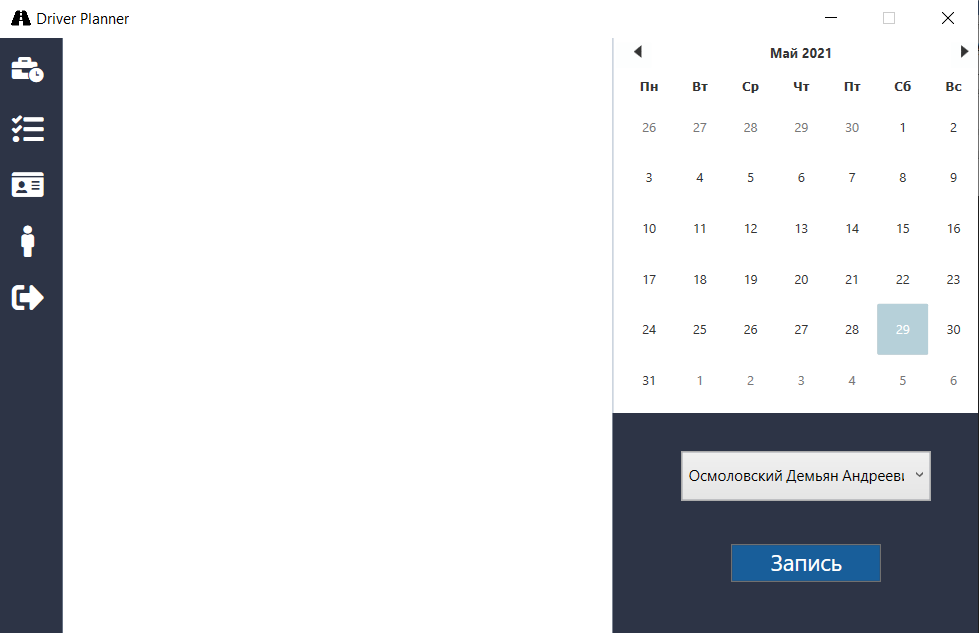


Рис. 4.10 – Выбор даты проведения занятия

После выбора даты клиент вызывает метод сервиса, который генерирует и возвращает коллекцию со свободными занятиями на заданный день, после чего клиент обрабатывает эти данные Рис. 4.11

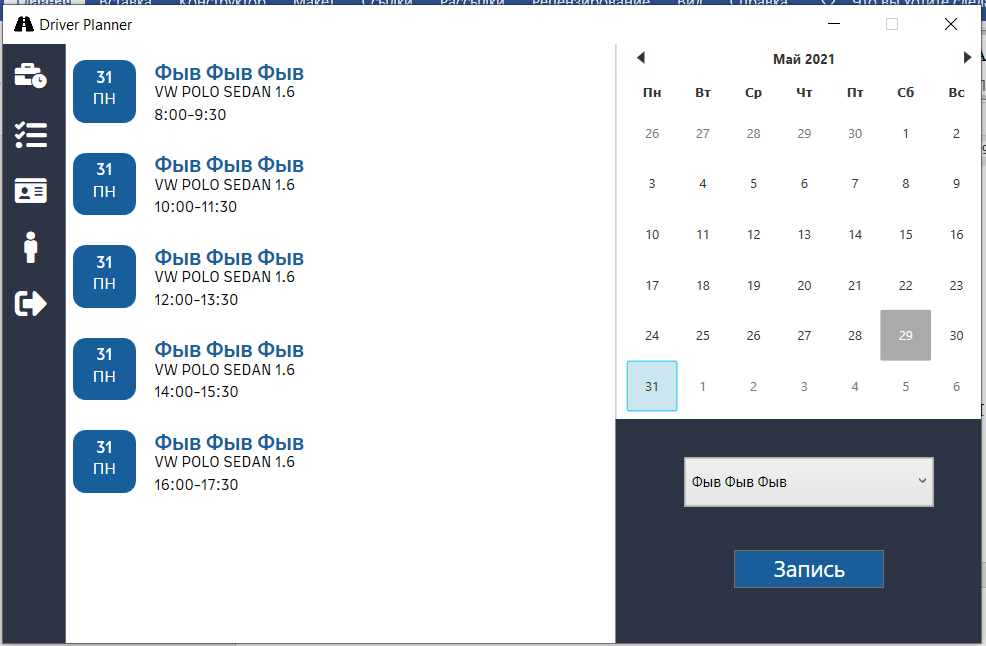


Рис. 4.11 – Выбор занятия

Генерация занятия производится на сервисе в методе GetClasses листинг – 4.10

|  |
| --- |
| public List<TimeTable> GetClasses(DateTime dateClass, int InstructorID)  {  List<TimeTable> defSet = new List<TimeTable>();  #region Check for weekend  switch (dateClass.DayOfWeek)  {  case DayOfWeek.Sunday:  case DayOfWeek.Saturday:  return defSet;  case DayOfWeek.Monday:  break;  case DayOfWeek.Tuesday:  break;  case DayOfWeek.Wednesday:  break;  case DayOfWeek.Thursday:  break;  case DayOfWeek.Friday:  break;  default:  break;  }  #endregion  using (var ctx = new DriverPlanner\_DB())  {  #region DefSet    var dbSet = ctx.TIMETABLE.Where(t => t.DateOfClass == dateClass && t.InstructorID == InstructorID).Include(t => t.Instructor.Car).Include(t=>t.ClassInterval).Include(t=>t.User.Gender).Include(t=>t.Instructor.Gender).ToList();    for (int i = 1; i <= ctx.INTERVALS.Count(); i++)  {  bool toAdd = true;  TimeTable tt = new TimeTable();  tt.ClassInterval = ctx.INTERVALS.Find(i);  tt.IntervalCode = i;  tt.DateOfClass = dateClass;  tt.InstructorID = InstructorID;  tt.Instructor = ctx.INSTRUCTORS.Where(t=>t.InstructorID == InstructorID).Include(t => t.Car).Include(t=>t.Gender).FirstOrDefault();  foreach (var item in dbSet)  {  if (item.ClassInterval.IntervalNumber == i)  {  toAdd = false;  }  }  if(toAdd)defSet.Add(tt);  }  return defSet;  #endregion  }  } |

Листинг 4.9 – Метод генерации объектов занятий

После выбора занятия останется только нажать кнопку записи и занятие забронируется за пользователем и он сможет его найти в специальной вкладке Рис 4.12

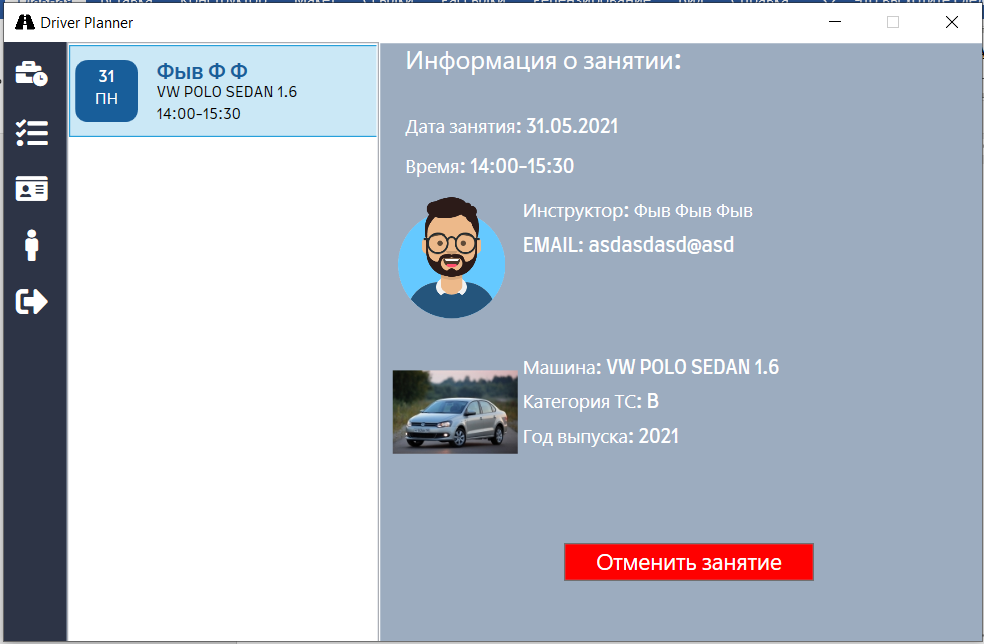


Рис. 4.12 – Выбор занятия

Если по каким-либо причинам занятие требуется отменить, то следует нажать кнопку отмены и клиент отправить запрос на сервис с целью отменить занятие.

# Тестирование, проверка работоспособности и анализ полученных данных

Протестируем приложение начиная с регистрации пользователя.

Регистрация пользователя производится в отдельном окне. Окно регистрации изображено на рисунке 5.1

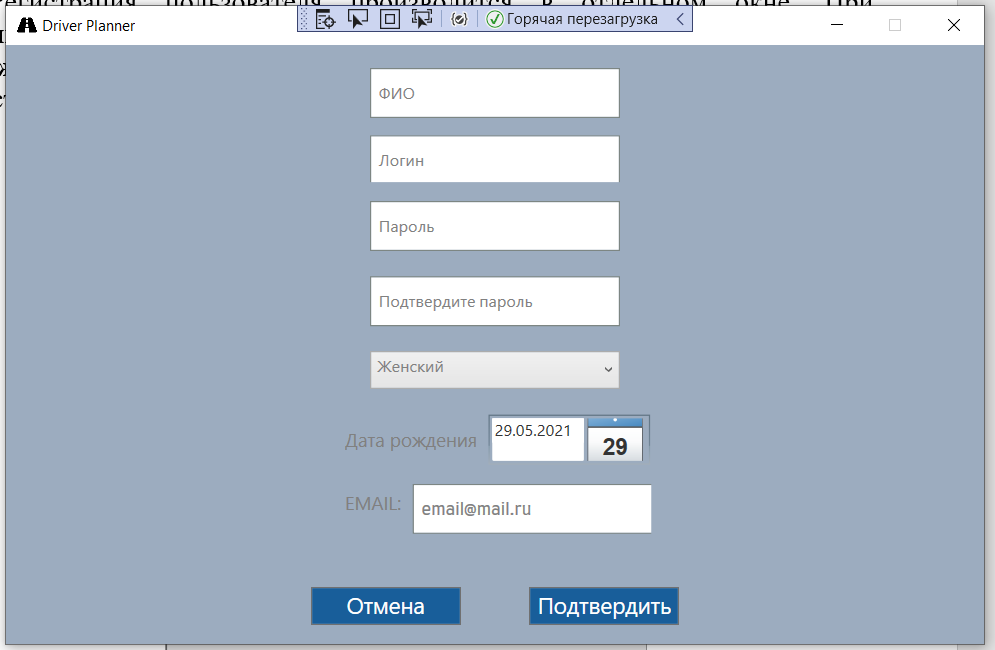


Рисунок 5.1 – Регистрация пользователя

При заполнении полей, они проверяются на валидность с помощью регулярных выражений и, при неверном вводе выводит ошибку в TextBox.

При попытке создать пользователя с ошибками появляется окно с подсказкой. Окно показано на рисунке 5.2

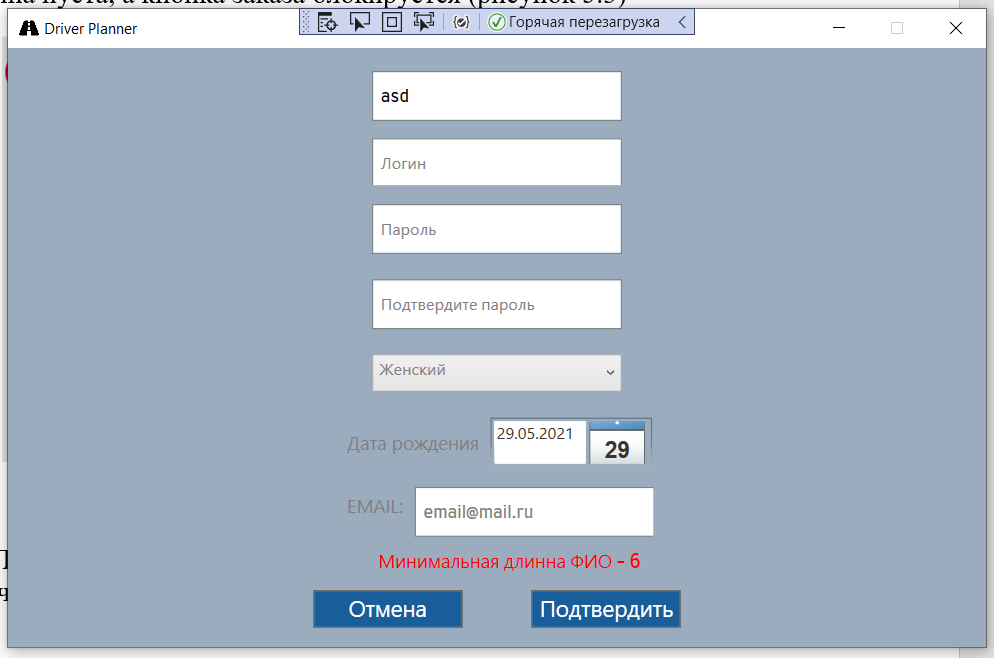


Рисунок 5.2 – Сообщение об ошибке

Далее войдем под созданным пользователем и попробуем сделать бронь. Просмотрим расписание и увидим, что все занятия на данное число свободны (рисунок 5.3)

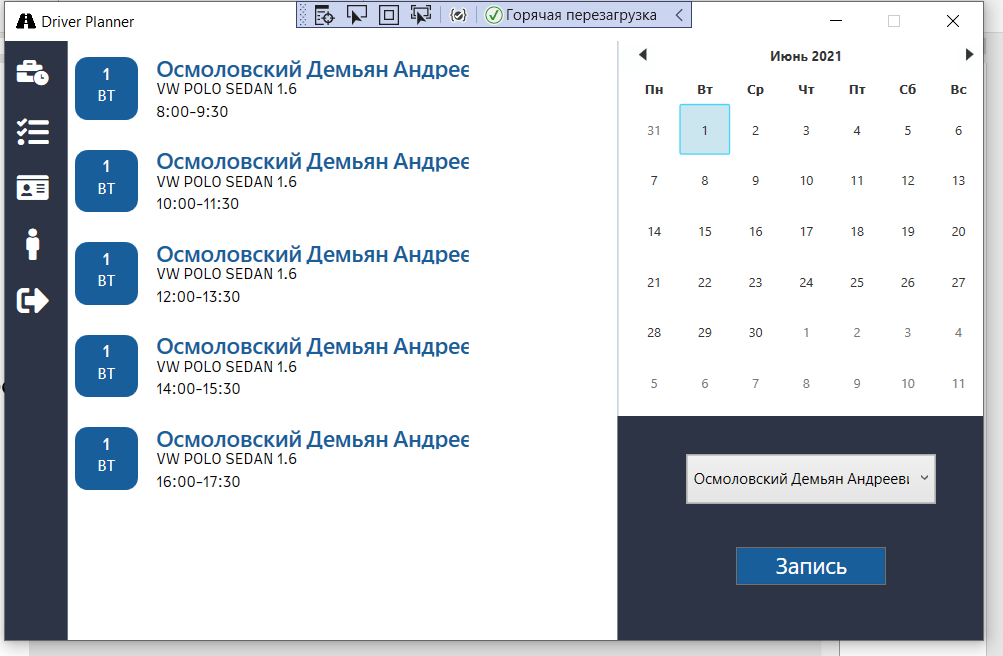


Рисунок 5.3 – Пустое окно заказа

Забронируем занятие. И найдем его в списке запланированных (рисунок 5.4).

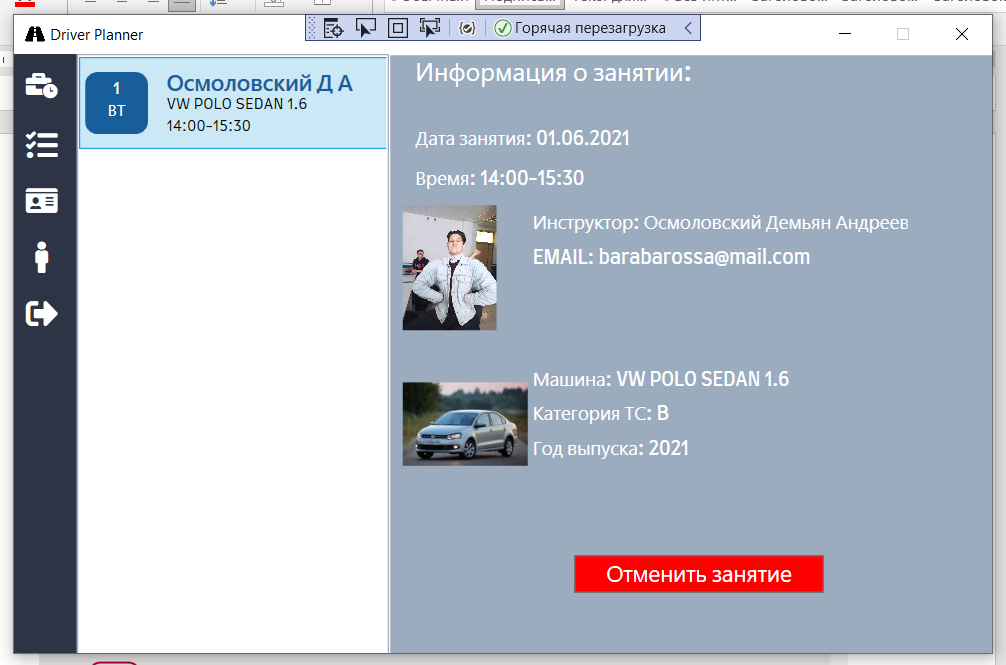


Рисунок 5.4 – Запланированные занятия

После успешного бронирования мы можем отслеживать все занятия во вкладке “Мои занятия” (рисунок 5.4).

Зайдём в систему под аккаунтом администратора, ему доступны следующие меню (рисунок 5.5)

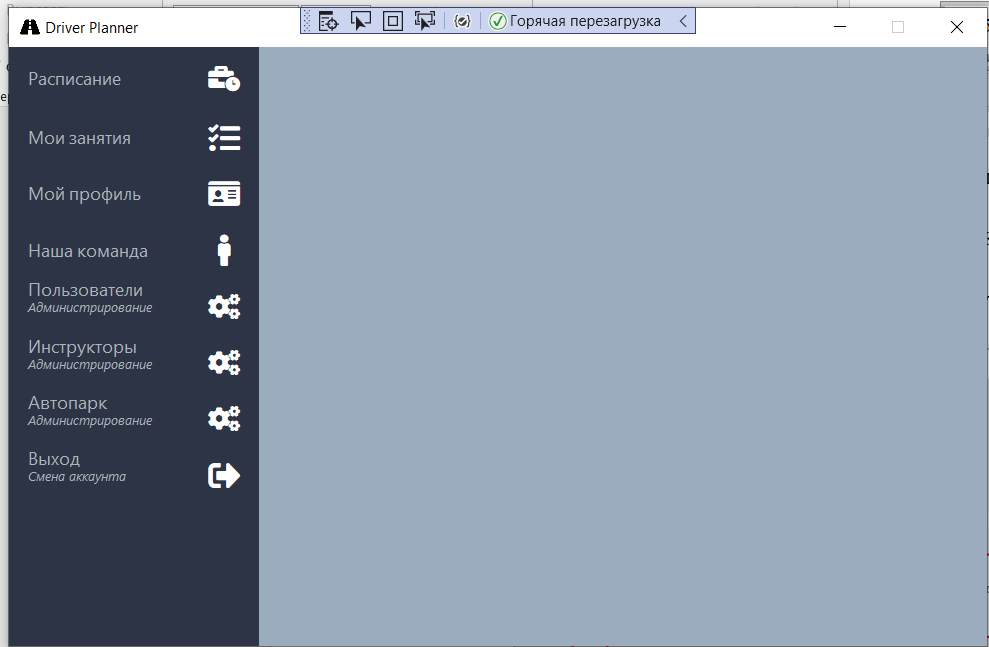


Рисунок 5.5 – Заказы пользователя

Администратор может просматривать проили других пользователей, а так же производить с ними определенные действия(Рис 5.6).

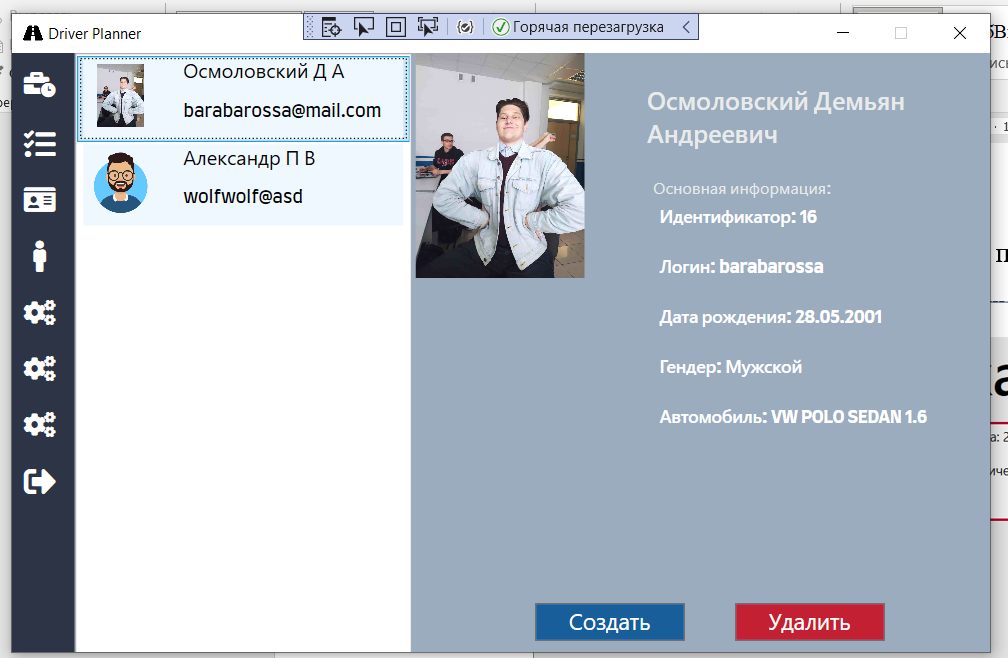


Рисунок 5.6 – Полное описание заказа

Войдем в качестве инструктора и проверим работу его функций.

Администратор может просматривать только свои занятия и может изменять свой профиль (рисунок 5.8-5.9).

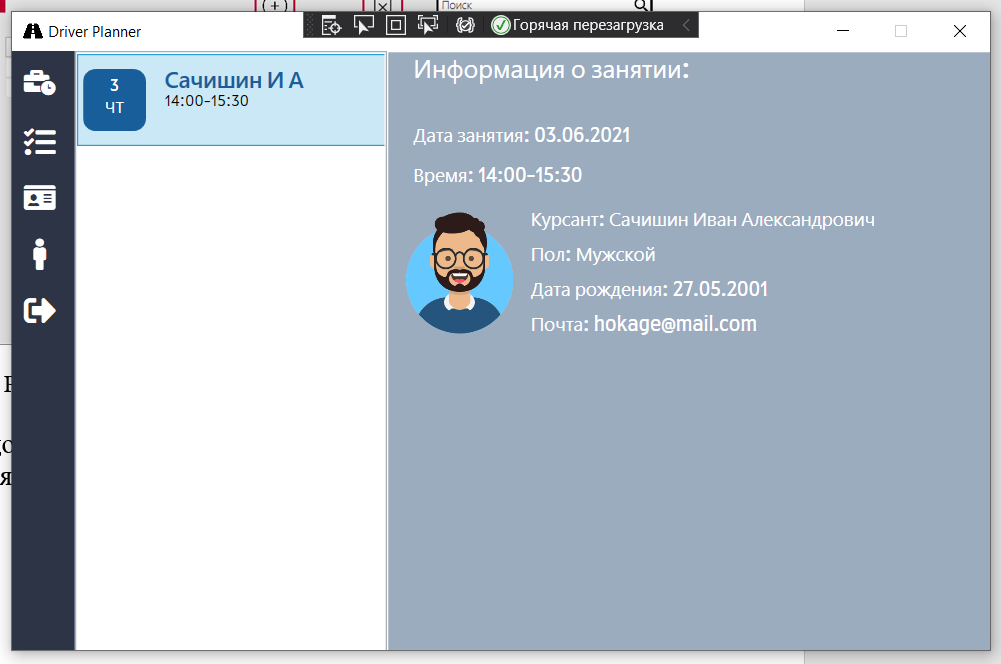


Рисунок 5.8 – Просмотр занятий инструктора

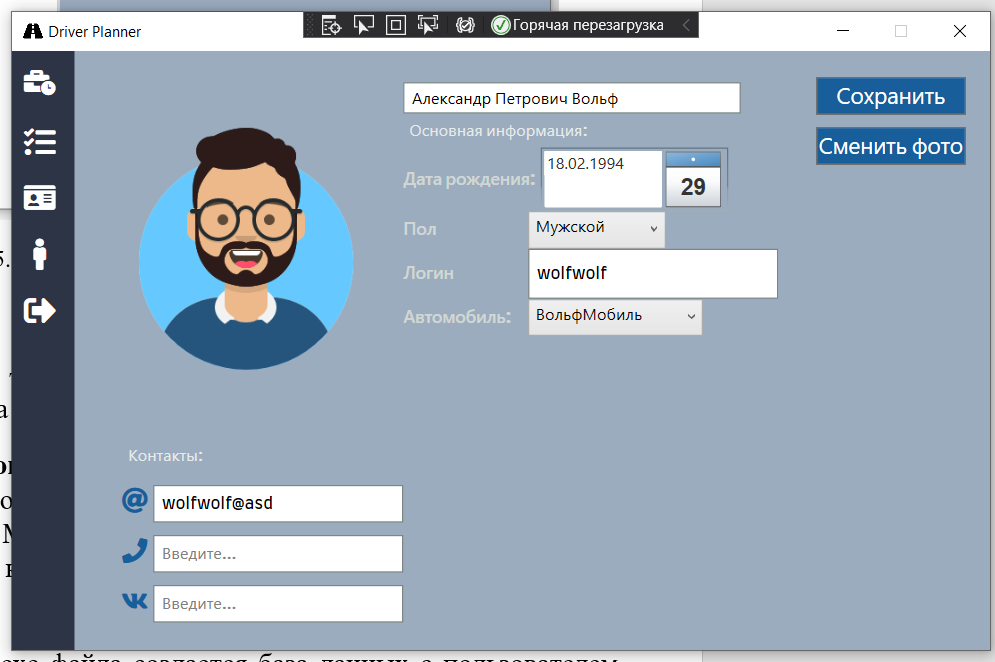


Рисунок 5.9 – Редактор профиля инструктора

Благодаря конвертеру, который преобразует индекс картинки в картинку, при нулевом индексе отображается стандартная картинка.

**Руководство по установке и использованию**

Для использования программного обеспечения потребуется:

* Установленный Microsoft SQL Server;
* Подключение к интернету (для связи с хостом).

При первом запуске .exe файла создается база данных с пользователем admino и паролем dominos. Этот пользователь имеет права администратора. Далее необходимо заполнить базу данных необходимой информацией, создав их из вкладки «Редактор автопарка» (рисунок 6.1 – 6.2).

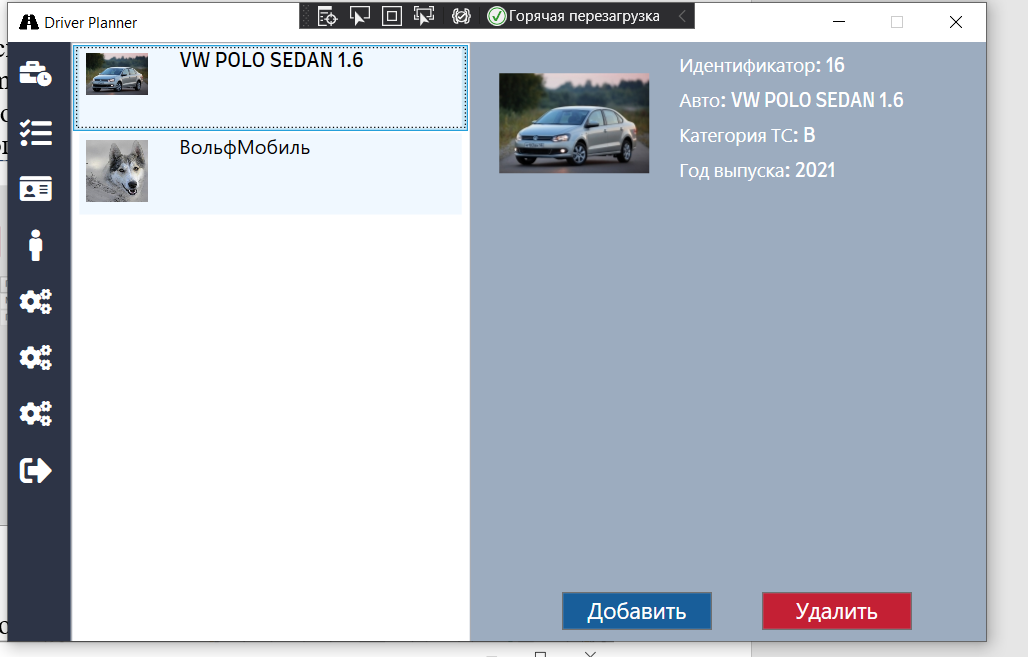


Рисунок 6.1 Редактор автопарка

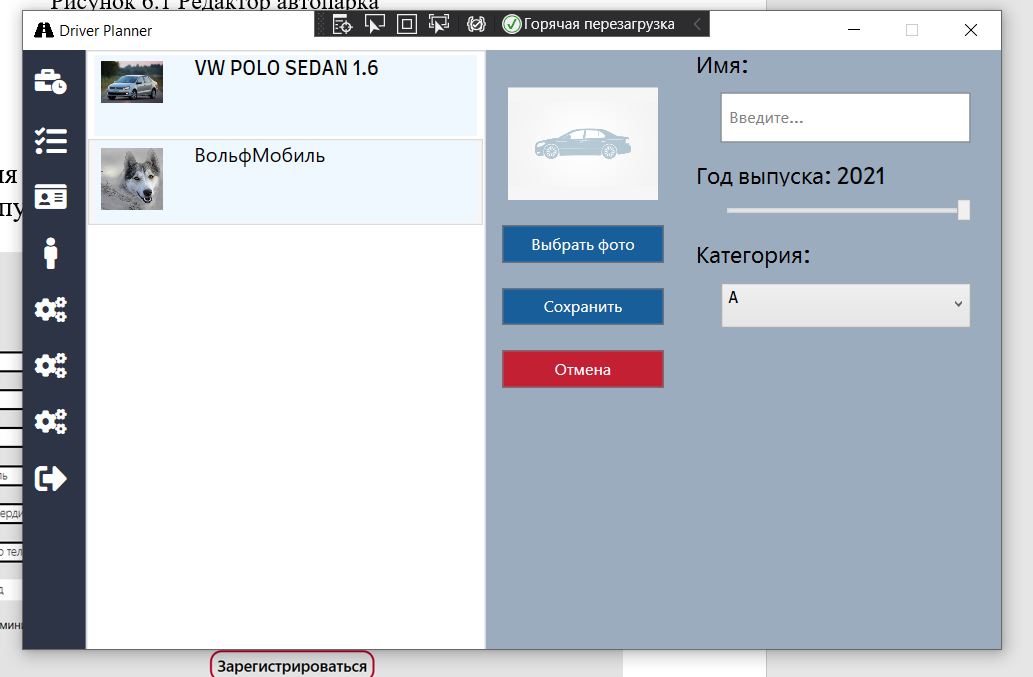


Рисунок 6.2 – Добавление авто

Удалять авто не получится до тех пор, пока инструктор не поменяет активную машину на другую.

# ЗАКЛЮЧЕНИЕ

При выполнении данного курсового проекта было разработано программное средство для автошкол «Driver Planner». Был проведен анализ аналогов для выявления сильных и слабых сторон других программных средств и разработки функциональных требований, проектирование и разработка приложения. Приложение было протестировано, а также проверено на полноту выполняемых операций.

Ошибок на стадии разработки выявлено не было, так как приложение работает без ошибок и соответствует всем поставленным требованиям.

# СПИСОК ЛИТЕРАТУРЫ

[1] METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com>. Дата доступа: 24.04.2021

[2] refactoring.guru Сайт о программировании [Электронный ресурс] / Режим доступа: <https://refactoring.guru>. Дата доступа: 24.04.2021

[3] docs.microsoft.com Сайт о программировании [Электронный ресурс] / Режим доступа: <https://docs.microsoft.com>. Дата доступа: 24.04.2021

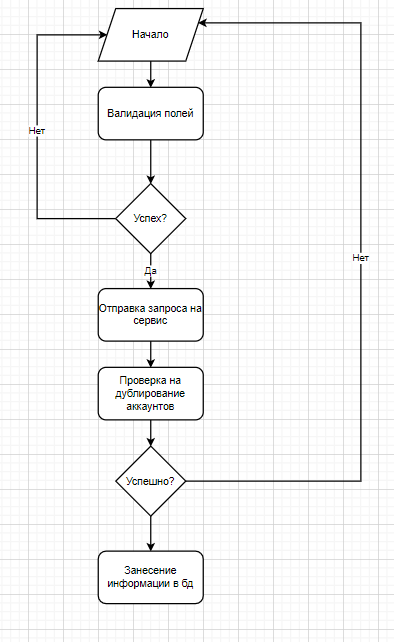
[4] http://dscontrol.ru/onlinelearning/ Сайт по продаже автозапчастей [Электронный ресурс] / Режим доступа: <http://dscontrol.ru/onlinelearning/> Дата доступа: 24.04.2021

[5] Википедия [Электронный ресурс] / Режим доступа: <https://ru.wikipedia.org/wiki/C_Sharp> Дата доступа: 24.04.2021

[7] Википедия [Электронный ресурс] / Режим доступа: <https://flexberry.github.io/ru/gpg_class-diagram.html> Дата доступа: 24.04.2021

# ПРИЛОЖЕНИЕ 1

Схема работы регистрации



# ПРИЛОЖЕНИЕ 2

Схема работы авторизации



# ПРИЛОЖЕНИЕ 3

Команда просмотра расписания

|  |
| --- |
| class UpdateViewCommand : ICommand  {  public event EventHandler CanExecuteChanged;    private MainViewModel \_viewModel;    public UpdateViewCommand(MainViewModel viewModel)  {  this.\_viewModel = viewModel;  }    public bool CanExecute(object parameter) => true;    public void Execute(object parameter)  {  if (parameter.ToString() == "Login")  {  \_viewModel.CurrentViewModel = new LoginViewModel(\_viewModel);  }  else if (parameter.ToString() == "Register")  {  \_viewModel.CurrentViewModel = new RegisterViewModel(\_viewModel);  }  else if (parameter.ToString() == "WorkingSpace")  {  \_viewModel.CurrentViewModel = new WorkingViewModel(\_viewModel);  }      }  } |